METHOD AND APPARATUS FOR PERFORMING MODULAR ARITHMETIC

BACKGROUND OF THE INVENTION

Field of the Invention

Many electronic interactions require the provision of a certain level of security to ensure that the data contained in a message transfer is difficult to intercept and decode, or it is capable of being verified as being genuine, or both. To achieve these ends, it is possible to encrypt data according to one of many possible schemes. A popular scheme is called public key cryptography (*e.g.*, PGP). Public key cryptography enables a particular message to be encoded according to an individual's private key and a third party's public key - both are long fixed numbers. The message may then be decoded by the third party through use of their private key. In this way, each party may keep their private key secret and thus control who is able to receive and decode any given message.

One of the key elements of encryption systems is the ability to be able to perform modular arithmetic. The basic calculation which is performed may be written as :

$$S = AB \bmod N \qquad (1)$$

where A, B and N are large numbers, typically including many hundreds of digits.

Cryptography systems are generally mathematically complex and can pose a high computational overhead on any system which implements them.

Description of the Related Art

Prior art systems for performing modular arithmetic make use of Montgomery's theorem, which has been used in many software and hardware implementations of modular arithmetic algorithms. Implementations using

Montgomery's theorem are able to compute a value for S without first multiplying A and B and then dividing by N. Most of the hardware implementations rely on an iterative approach which decomposes A into k blocks of p bits to limit the size of the hardware operators required. Further advances have used a serial architecture to further reduce the circuit size. Such architectures are generally based around two serial multipliers, FIFO elements, and the pre-computation of a constant J_0, such that :

$$J_0.N \equiv -1 \bmod 2^p \qquad (2)$$

k and p are both positive integers, and the binary representation of a positive integer X, where $X < 2^{kp}$ may be given by:

$$X = \sum_{i=0}^{k-1} X[i]2^i \qquad (3)$$

where $0 \leq X[i] < 2$, *i.e.*, X may be either 0 or 1.

Throughout this specification, square brackets [ ] refer to a particular bit position in a multi-bit word *e.g.*, $X[i]$ refers to the i^{th} bit of word X. Angle brackets < > refer to a particular block of a multi-bit word *e.g.*, $X\langle i\rangle$ refers to the i^{th} block of word X. Parentheses ( ) refer to the value of a word at a particular iteration of a loop function *e.g.*, $X(i)$ refers to the value of word X at the i^{th} iteration.

A definition for $X[j:k]$, where $j > k$, is that X is a positive integer having a total length of $j+1-k$ bits, such that $X[j]$ is the MSB and $X[k]$ is the LSB.

The base 2^p representation of X is given by:

$$X = \sum_{i=0}^{k-1} X\langle i\rangle 2^{pi} \qquad (4)$$

where $0 \leq X\langle i\rangle < 2^p$

In the following description, it is assumed that N is an odd integer such that $2^{p(k-1)} < N < 2^{kp}$, and that both A and B are less than N. A p-bit constant, J_0, is thus defined as:

$$J_0.N<0> \equiv -1.\mathrm{mod}2^p \qquad (5)$$

N is the modulus number which is used in all public key cryptography systems. It is defined as the product of two large prime numbers (*i.e.*, >>2) and must therefore be odd.

The prior art hardware implementation of the Montgomery theorem may be described by the following pseudo-code.

```
1.  procedure MM-BASIC(A,B,N)
2.  S(-1) = 0
3.  for i=0 to k-1
4.      T = S(i - 1) + A<i>B
5.      Y0 = (T.J0) mod 2^p
6.      S(i) = (T + NY0)/2^p
7.      if S(i) ≥ N then S(i) = S(i) - N
8.  end for
```

The implementation of this pseudo code in hardware is shown in a simplified form in Figure 1. The architecture is constructed in serial form so that one bit of the solution is generated for each clock cycle. Such an architecture, as opposed to a parallel one, minimizes the amount of hardware required at the expense of speed.

The circuit of figure 1 is arranged to receive five different input signals: $A[k]$ 200; $B[t]$ 205; $S(i\text{-}1)$ 210; $GE(i\text{-}1)$ 215; and $N[t]$ 220.

Serial Multiplier 110 accepts as inputs, a fixed p-bit word, $A<i>$ produced by register 105, and a one-bit data stream $B[t]$ 205. It then acts to produce the output, $(A<i>.B)$, one bit at a time.

Multiplier 110 is configured internally as shown in Figure 2. The two inputs are the output 340 of register 105 and $B[t]$ 205. The two inputs 205, 340 are ANDed together in AND gate 300. The result of this operation is fed into Carry Save Adder 310, along with two other inputs. The first of these other inputs is the carry output (C) derived from the fed back output from p-bit register 315. The other input to the Adder is derived from the result output (R) of p-bit register 320 which has been divided by 2 in divider 305. Registers 315, 320 are positioned immediately after the Carry Save Adder 310 and each receives one of the twin outputs produced by the adder.

The Carry Save Adder 310 is arranged to transform a sum of three numbers into a sum of two numbers such that:

$$2.C + R = X + Y + Z \tag{6}$$

The Carry Save Adder 310 computes $C(t)$ and $R(t)$ based on the following bitwise Boolean equations.

$$C(t) = (C(t - 1) \text{ OR } R(t - 1)/2) \text{ AND } (C(t - 1) \text{ AND } B[t].A<i>) \text{ AND } (R(t - 1)/2 \text{ AND } b[T].A<i>) \tag{7}$$

$$R(t) = C(t - 1) \oplus R[t].A<i> \tag{8}$$

In a simplified notation:

$$C(t), R(t) = \text{SERIAL\_MULT } (B[t].A<i>, C(t\text{-}1), R(t\text{-}1)) \tag{9}$$

The procedure MM_BASIC, already shown, may be written in a form which shows the serial operations explicitly:

```
1.  procedure MM-SERIAL(A, B, N)
2.  S(-1) = 0
3.  GE(-1) = 0
4.  for i=0 to k-1
5.      #computation of Y0
6.      for t= 0 to p-1
7.          CS1(t), RS1(t) = SERIAL_SUB(CS1(t-1), GE(i-1) . N[t], S(i-1)[t])
8.          CM1(t), RM1(t) = SERIAL_MULT(B[t] . A<i>, CM1(t-1), RM1(t-1))
9.          CA1(t), RA1(t) = SERIAL_ADD(CA1(t-1), RM1(t)[0], RS1(t))
10.         CM2(t), RM2(t) = SERIAL_MULT(RA1(t) . J0, CM2(t-1), RM2(t-1))
11.     Y0[t] = RM2(t)
12.     end for
13.     # mail loop: computation of S(i)
14.     for t = 0 to kp + p-1
15.         CS1(t), RS1(t) = SERIAL_SUB(CS1(t-1), GE(i-1) . N[t], S(i-1)[t])
16.         CM1(t), RM1(t) = SERIAL_MULT(B[t] . A<i>, CM1(t-1), RM1(t-1))
17.         CA1(t), RA1(t) = SERIAL_ADD(CA1(t-1), RM1(t)[0], RS1(t))
18.         CM2(t), RM2(t) = SERIAL_MULT(RA1(t) . J0, CM2(t-1), RM2(t-1))
19.         CA2(t), RA2(t) = SERIAL_ADD(CA1(t-1), RM2(t)[0], RA1(t))
20.         S(i)[t-p] = RA2(t)
21.         SGE(t) = SERIAL_GE(SGE(t-1), N[t-p], S(i)[t-p])
22.     end for
23.     GE(i) = SGE(kp+p-1)
24. end for
```

The total number of clock cycles required to compute the result according to the above scheme is $k(kp+2p)$.

BRIEF SUMMARY OF THE INVENTION

In accordance with one embodiment of the present invention, an apparatus is provided that includes inputs A, B and N, and an output S, said apparatus being arranged to perform a modular operation, S=A.B mod N, the apparatus including a 2-stage Carry Save Adder (2-CSA) and a 1-stage Carry Save Adder (1-CSA), the 2-CSA being arranged to receive 5 input signals: U_0, being the partial product of N and Y_0; U_1, being the subtraction of a previous version of S and U_6 wherein U_6 is either N or 0 depending on the value of the comparison between the result of the previous iteration and N; U_2, being the partial product of B with the current version of A; U_3, being S/2; and U_4, being the carry output of the 1-CSA; where result and carry outputs of the 2-CSA form two of three inputs to the 1-CSA, wherein the result (R) output of the 1-CSA is the desired result (S), and the third input to the 1-CSA is a compensation signal arranged to allow S to be calculated without knowing the constant J_0, where $J_0N<0> = -1. \mod 2^p$, where p is a block length into which A is sub-divided.

In a second broad form, an embodiment of the present invention provides An iterative method of performing a modular operation of S = A.B mod N, where A, B and N are encoded as multi-bit digital words, including the following steps: a) setting S(-1) to 0, and i to 0; b) setting S(i) to $(S(i-1) + A<i>B + NY_0)/2^p$; c) setting S(i) to (S(i) - N) if $S(i) \geq N$; d) repeating steps b) and c) k times, wherein: i is a loop counter; k is a number of blocks of p bits length into which A is divided; $Y_0 = ((T.J_0) \mod 2^p)$; $J_0N = -1 \mod 2^p$; and Y_0 is calculated one bit at a time, based on the fact that $(T + NY_0)$ is a multiple of 2^p.

In accordance with another embodiment of the invention, an apparatus for performing modular arithmetic is provided, the apparatus includes a first AND gate configured to receive first and second inputs and to generate a first output; a second AND gate configured to receive third and fourth inputs and to generate a second output; a divider configured to generate a third output; a first carry-save adder configured to receive as inputs the first output from the first AND

gate, the second output from the second AND gate, and the third output from the divider and to generate fourth and fifth outputs; a second carry-save adder configured to receive the combination of the fourth output and a fifth input as one input and to receive the fifth output as a second input and to generate a carry output that is fed back into a third input of the second carry-save adder and to generate a result output that is an input to the divider and the desired result.

In accordance with yet a further embodiment of the invention, an apparatus for performing modular arithmetic is provided, the apparatus having inputs A, B, and N and an output S; a first carry-save adder configured to receive five input signals that include: U_0, the partial product of N and Y_0, where Y_0 equal $((T.J_0) \bmod 2^p)$; U_1, the subtraction of a previous version of S and U_6, wherein U_6 is one of N or 0 depending on the value of a comparison between a result of a previous iteration and N; U_2, a partial product of input B and a current version of input A; U_3, the result of S/2; and U_4, a carry output of a second carry-save adder; the first carry-save adder configured to generate a result output and a carry output; the second carry-save adder configured to receive the result output and the carry output from the first carry-save adder and to receive a compensation signal as a third input and to generate a desired result and the carry output U_4; and wherein $J_0N<0> = -1. \bmod 2^p$, where p is a block length into which A is sub-divided.

Other features and benefits of the invention will become apparent in the following description of various embodiments of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

For a better understanding of the present invention and to understand how the same may be brought into effect, the invention will now be described by way of example only, with reference to the appended drawings in which:

Figure 1 shows a simplified prior art circuit for implementing modular arithmetic according to Montgomery's theorem;

Figure 2 shows a prior art serial/parallel multiplier or carry save adder;

Figure 3 shows a merged multiplier as used in embodiments of the invention; and

Figure 4 shows a hardware implementation according to an embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

The present invention retains a serial architecture to accomplish the calculation, but embodiments of the inventions do not require pre-knowledge of the constant, J_0. Embodiments of the invention calculate $Y_0 = ((T \cdot J_0) \bmod 2^p)$ one bit at a time, based on the fact that $(T + NY_0)$ must be a multiple of 2^p. In this way, the complex mathematical functions required to pre-compute J_0 can be dispensed with.

With this implicit knowledge, the procedure MM-BASIC described previously, may now be written as MM-SIMPLE:

1. **procedure** MM-SIMPLE(A, B, N)
2. $S(-1) = 0$
3. **for** $i = 0$ **to** k-1
4. $\quad S(i) = (S(i\text{-}1) + A\langle i \rangle B + NY_0)/2^p$
5. $\quad$ **if** $S(i) \geq N$ **then** $S(i) = S(i) - N$
6. **end for**

The above serial implementation of MM-SIMPLE is more efficient than the prior art implementation of MM-BASIC that has two multipliers required instead of the single multiplier in the disclosed embodiments of the invention. The gain, in terms of fewer components, is a total of $2p$ registers plus the two serial adders 120 and 155. The removal of the need for these components removes a significant amount of circuitry, and thus the resulting architecture requires less

space and consumes less power to achieve the same result. This design also calculates the result in fewer clock cycles.

Figure 3 shows the resultant hardware implementation which may be used to perform the steps of procedure MM-SIMPLE presented above. As shown therein, an apparatus for performing modular arithmetic is provided that includes a first AND gate 400 receiving inputs 470 and 475 and generating a first output, a second AND gate 410 receiving as inputs 480 and 485 and generating a second output, and a divider 420 generating a cleared output. A first carry-save adder 430 receives on a first input the first output of the first AND gate 400, on a second input the second output of the second AND gate 410, and on a third input an output of the divider 420, and generates therefrom a first output and a second output that are received as first and second inputs of a second carry-save adder 440. It is to be noted that the first output of the first carry-save adder 430 is combined with a fifth input 465 prior to being received at the second carry-save adder 440. The second carry-save adder 440 generates on a first output a carry out that is received at a first register 450, the output of which becomes a third input to the second carry-save adder 440. A second output of the carry-save adder 440 is received at a second register 460 and becomes the desired result 490 that is also the input to the divider 420.

Y_0 is computed bit by bit during the first p cycles of the loop, starting at line 15 of the procedure MM-SERIAL. Assuming that at cycle $q<p$, the bits 0, 1,...q-1 have already been computed, leaving only bit q to be discovered.

According to embodiments of the present invention, if, at cycle q, the LSB of the 2-stage Carry Save Adder shown in Figure 3 is '1', then $N[q:0]$ is added to the intermediate result, and $Y_0[q] = 1$.

This may be proved as follows. At the q^{th} step, the intermediate values from the first Carry Save Adder may be given as

$$S = 2C + R \tag{10}$$

$$=(A<i>.B[q:0]+Y_0[q-1:0].N[q:0]+S<i-1>[q:0])/2^q \quad (11)$$

Assuming that the q^{th} bit of Y_0 is a '1', then the above equation may be re-written as:

$$S'=(A<i>.B[q:0] + (2^q + Y_0[q-1:0]).N[q:0] + S<i-1>[q:0])/2^q \quad (12)$$

$$= S + N[q:0] \quad (13)$$

As the LSB of N is always 1, since it is a large prime number and, therefore, odd, then from the above equations, it can be seen that the LSBs of S and S′ are always inverted. Therefore, it is possible to guarantee that the LSB of the result is 0 in the first p steps by choosing either S or S′. The choice of S′ implies that the q^{th} bit of Y_0 must be forced to equal 1.

The above step is repeated at each cycle q<p, so that at the end all bits of Y_0 are discovered.

The procedure, MM-SERIAL-SIMPLE shown below is a pseudo-code implementation of an embodiment of the present invention, and is a version of the previously presented MM-SERIAL adapted according to the above results.

```
1.  procedure MM-SERIAL-SIMPLE(A, B, N)
2.  S(-1) = 0
3.  GE(-1) = 0
4.  for i = 0 to k-1
5.      # main loop: computation of S(i)
6.      for t = 0 to kp+p-1
7.          C_S1(t), R_S1(t) = SERIAL_SUB(C_S1(t-1), GE(i-1) . N[t], S(i-1)[t])
8.  C_int, R_int = 2-STAGE_CSA(B[t] . A<i>, C_M(t-1), R_M(t-1)/2,
                       N[t] . Y_0)
9.          if t < p and R_int[0] = 1 then
```

```
10.                 C_M(t), R_M(t) = CSA(N[t:0], C_int, R_int)
11.                 Y_0[f] = 1
12.             else
13.                 C_M(t), R_M(t) = C_int, R_int
14.             end if
15.             S(i)[t-p] = R_M(t)[0]
16.             SGE(t) = SERIAL_GE(SGE(t-1), N[t-p], S(i)[t-p])
17.         end for
18.         GE(i) = SGE(kp+p-1)
19.     end for
```

The conditional statement at line 9 of the above procedure may be considered to trigger a compensation event which, if t<p and $R_{int}[0]$ =1, causes the value of register 525 N_{del} to be applied to the input of the 1-stage CSA (1-CSA) 540. If the condition is not satisfied, then the C and R outputs of the 2-stage CSA (2-CSA) 520 merely feed straight into the 1-CSA and no compensation is performed.

It is the addition of the compensation function that directly removes the need to explicitly compute J_0.

In figure 4, the compensation function is implemented by register 525, AND gate 530, MUX 535. The MUX 535 effectively performs the conditional IF statement of line 9 of MM-SERIAL-SIMPLE, and if $R_{int}[0]$ is equal to 1, then the contents of register 525 is applied to 1-CSA 540.

The above procedure (MM-SERIAL-SIMPLE) is further explained in the procedure below (MM-SERIAL-SIMPLE_enhanced), which includes further details on selected ones of the internal signal nets.

These internal nets are labelled from U_0 to U_8 and directly correspond with selected internal nets shown in Figure 4.

```
1.  procedure MM-SERIAL-SIMPLE_enhanced(A, B, N)
```

2. S(-1) = 0
3. GE(-1) = 0
4. A_{next} = A[p-1:0]
5. **for** i=0 **to** k-1
6. # main loop: computation of $S(i)$
7. N_{del} = 0
8. Y_0 = 0
9. R = 0
10. C = 0
11. $A_{current}$ = A_{next}
12. A_{next} = A[(i+1)(p-1):(i+1)p]
13. **for** t = 0 **to** kp+p-1
14. U_0 = AND2(N[t], Y_0)
15. U_6 = AND1(GE(i-1), N[t])
16. U_1 = SUB1(U_6, S(i-1[t])
17. U_2 = AND3(B[t], $A_{current}$)
18. U_3 = R/2
19. U_4 = C
20. C_{int}, R_{int} = 2-STAGE-CSA(U_0, U_1, U_2, U_3, U_4)
21. U_7 = MUX(R_{int}[0],0)
22. U_5 = AND4(U_7, N_{del})
23. **if** t < p **then**
24. Y_0[t] = U_7
25. N_{del}[t] = N[t]
26. $U8$=0
27. **else**
28. # N_{del} acts as a shift register
29. U_8 = N_{del}[0]
30. N_{del} = N_{del}/2
31. N_{del}[p-1] = N[t]
32. **endif**

33. $C,R = \mathrm{CSA}(U_5, C_{int}, R_{int})$
34. $S(i)[t] = R[0]$
35. $SGE(t) = GE(U_8, R[0])$
36. **end for**
37. $GE(i) = SGE(kp+p-1)$
38. **end for**

As an example, presented below are details of how an embodiment of the invention operates on some sample input data. The following inputs are provided in 32-bit format:

$A = C7197F0E$

$B = CCEFBAE4\_77AF9EE5\_848D8AE6$

$N = D077EC53\_F4AA27A4\_D7816723$

The result of the Montgomery multiplication of A by B is given by $(AB+NY_0)/2^p$. Before the computation starts, the registers of the multiplier are initialized as follows.

$N_0 = 00000003$

$Y_0 = 00000000$

$RC = 0\_00000000$

$B[t] = 6$

$N[t] = 3$

For the sake of simplicity, the registers R and C have been summed into register RC, and the computation is performed 4 bits (a nibble) at the time, thus setting p=4.

1. Computation of the intermediate results, based on the partial products

$N[t].Y0$	= 0_00000000
$+B[t].A$	= 4_*AA98FA54*
$+RC/16$	= 0_00000000
= *Intermediate*	= 4_*AA98FA54*

2. Find the first 4 bits of compensation value (Z) such that the 4 LSBs of *Intermediate*$+Z.N_0$ are all zero.

Z	= 4

5 3. Add the partial product $Z.N_0$ to *Intermediate*

Intermediate	= 4_*AA98FA54*
$+Z.N_0$	= 0_0000000*C*
$=RC$	= 4_*AA98FA60*

4. Update the registers with- the new values and restart the cycle

N_0 = 00000023 Y_0 = 00000004 RC = 4_*AA98FA60* $B[t] = E$ $N[t] = 2$

1. Computation of the intermediate results, based on the partial products

$N[t].Y0$	= 0_0000008*C*
$+B[t].A$	= *A*_*E*364*F*2*C*4
$+RC/16$	= 0_4*AA*98*FA*6
$=Intermediate$	= *B*_2*E*0*E*8272

2. Find first 4 bits of compensation (Z) such that the 4 lsb of $Intermediate+Z.N_0$ are all zero.

5 Z $=A$

3. Add the partial product $Z.N_0$ to *Intermediate*

Intermediate	= *B*_2*E*0*E*8272
$+Z.N_0$	= 0_0000015*E*
$=RC$	= *B*_2*E*0*E*83*D*0

4. Update the registers with the new values and restart the cycle

N_0 = 00000723 Y_0 = 000000*A*4 RC = *B*_2*E*0*E*83*D*0 $B[t] = A$ $N[t] = 7$

1. Computation of the intermediate results, based on the partial products

$N[t].Y0$	= 0_0000047*C*
$+B[t].A$	= 7_*C*6*FEF*68*C*
$+RC/16$	= 0_*B*2*E*0*E*83*D*
$=Intermediate$	= 8_79*DFE*345

2. Find first 4 bits of compensation (Z) such that the 4 lsb of $Intermediate+Z.N0$ are all zero.

Z =9

3. Add the partial product $Z.N_0$ to *Intermediate*

Intermediate	= 8_79*DFE*345
$+Z.N_0$	= 0_0000403*B*
$=SUM_2$	= 8_79*E*02380

4. Update the registers with the new values and restart the cycle

N_0 = 00006723 Y_0 = 000009*A*4.*RC* = 8_79E02380 $B[t]$ = 8 $N[t]$ = 6

This process is repeated until all the bits of Y_0 are discovered. At this stage, the compensation phase is no longer needed so the computation iterates over the remaining bits of *B* and *N*. The step by step result at each phase is given by the following table:

Cycle	N_0	Y_0	RC	$B[t]$	$N[t]$
0	XXXXXXXX	XXXXXXXX	XXXXXXXXXX	X	X
1	00000003	00000000	0000000000	6	3
2	00000023	00000004	04AA98FA60	E	2
3	00000723	000000A4	0B2E0E83D0	A	7
4	00006723	000009A4	0879E02380	8	6
5	00016723	000009A4	06C06A3480	D	1
6	00816723	000A09A4	0A88602800	8	8
7	07816723	000A09A4	06E1A24810	4	7
8	D7816723	090A09A4	03CE530470	8	D
9	D7816723	790A09A4	0CCFBD7800	5	4
10	4D781672	790A09A4	0694A37956	E	A
11	A4D78167	790A09A4	1007138AC1	E	7
12	7A4D7816	790A09A4	0F331C6EEC	9	2
13	27A4D781	790A09A4	08E52B51B4	F	A
14	A27A4D78	790A09A4	10F3358755	A	A
15	AA27A4D7	790A09A4	0D9096AF69	7	4
16	4AA27A4D	790A09A4	082EE40AE8	7	F
17	F4AA27A4	790A09A4	0D0C374AAC	4	3
18	3F4AA27A	790A09A4	0558478DCE	E	5
19	53F4AA27	790A09A4	0D961B9BD4	A	C
20	C53F4AA2	790A09A4	0E4CD923F9	B	E
21	EC53F4AA	790A09A4	1011728ED1	F	7
22	7EC53F4A	790A09A4	0FFADBDE3B	E	7
23	77EC53F4	790A09A4	0F3258F423	C	0
24	077EC53F	790A09A4	0A485783EA	C	D
25	D077EC53	790A09A4	101F39EA3A	0	0
26	0D077EC5	790A09A4	0101F39EA3	0	0
27	00D077EC	790A09A4	00101F39EA	0	0

Cycle	N_0	Y_0	*RC*	*B[t]*	*N[t]*
28	000D077E	790A09A4	000101F39E	0	0
29	0000D077	790A09A4	0000101F39	0	0
30	00000D07	790A09A4	00000101F3	0	0
31	000000D0	790A09A4	000000101F	0	0
32	0000000D	790A09A4	0000000101	0	0
33	00000000	790A09A4	0000000010	0	0
34	00000000	790A09A4	0000000001	0	0

Notice that the serial output result can be read directly as the right most nibble of the RC column. It is also interesting to notice the shifting pattern of N_0. From cycle 1 to 8, the register behavior is comparable to a stack, where the nibble are pushed from the left. From cycle 9 onward, the register behaves as a right shift register. The output of this register shall be used as the input of a comparator which detects if the results is greater or equal to *N*.

$$Y = 790A09A4$$
$$RESULT = 1\_01F39EA3\_AA38194E\_C8954C16\_00000000$$

In the light of the foregoing description, it will be clear to the skilled man that various modifications may be made within the scope of the invention.

The present invention includes any novel feature or combination of features disclosed herein either explicitly or any generalization thereof irrespective of whether or not it relates to the claimed invention or mitigates any or all of the problems addressed.

All of the above U.S. patents, U.S. patent application publications, U.S. patent applications, foreign patents, foreign patent applications and non-patent publications referred to in this specification and/or listed in the Application Data Sheet, are incorporated herein by reference, in their entirety.

From the foregoing it will be appreciated that, although specific embodiments of the invention have been described herein for purposes of illustration, various modifications may be made without deviating from the spirit and scope of the invention. Accordingly, the invention is not limited except as by the appended claims.